CLAIMS

[Claim(s)]

[Claim 1] The record medium which recorded the operating system equipped with the command which starts the thread manager which generates the thread which receives a demand message, and by which the processing based on the demand message concerned is made as an application program interface, the command which terminates a thread manager, and the command which registers into a thread manager the content processed by the thread and in which computer reading is possible.

[Claim 2] The above-mentioned operating system is a record medium which is characterized by having the command which delivers a demand message to a thread manager as an application program interface and in which computer reading according to claim 1 is possible.

[Claim 3] The above-mentioned operating system is a record medium which is characterized by having the command to which an answerback message is made to output from a thread manager, and the command which receives the answerback message from a thread manager as an application program interface and in which computer reading according

to claim 1 is possible.

[Claim 4] The above-mentioned operating system is a record medium which is characterized by having the command which deletes the content registered into the thread manager as an application program interface as a content processed by the thread and in which computer reading according to claim 1 is possible.

[Claim 5] The record medium which recorded the operating system equipped with the class including the procedure which starts the thread manager which generates the thread which receives a demand message, and by which the processing based on the demand message concerned is made, the procedure which terminates a thread manager, and the procedure which registers into a thread manager the content processed by the thread as an application program interface and in which computer reading is possible.

[Claim 6] The above-mentioned class is a record medium which is characterized by including the procedure which delivers a demand message to a thread manager and in which computer reading according to claim 5 is possible.

[Claim 7] The above-mentioned class is a record medium which is characterized by including the procedure to which an answerback message is made to output from a thread manager, and the procedure which receives the answerback message from a thread manager and in which computer reading according to claim 5 is possible.

[Claim 8] The above-mentioned class is a record medium which is characterized by including the procedure which deletes the content registered into the thread manager as a content processed by the thread and in which computer reading according to claim 5 is possible.

[Claim 9] The record medium which recorded the application program executed using the operating system equipped with the command which

starts the thread manager which generates the thread which receives a demand message, and by which the processing based on the demand message concerned is made, the command which terminates a thread manager, and the command which registers into a thread manager the content processed by the thread as an application program interface and in which computer reading is possible.

[Claim 10] The above-mentioned operating system is a record medium which is characterized by having the command which delivers a demand message to a thread manager as an application program interface and in which computer reading according to claim 9 is possible.

[Claim 11] The above-mentioned operating system is a record medium which is characterized by having the command to which an answerback message is made to output from a thread manager, and the command which receives the answerback message from a thread manager as an application program interface and in which computer reading according to claim 9 is possible.

[Claim 12] The above-mentioned operating system is a record medium which is characterized by having the command which deletes the content registered into the thread manager as an application program interface as a content processed by the thread and in which computer reading according to claim 9 is possible.

[Claim 13] The record medium which recorded the application program executed using the operating system equipped with the class including the procedure which starts the thread manager which generates the thread which receives a demand message, and by which the processing based on the demand message concerned is made, the procedure which terminates a thread manager, and the procedure which registers into a thread manager the content processed by the thread as an application program interface and in which computer reading is possible.

[Claim 14] The above-mentioned class is a record medium which is characterized by including the procedure which delivers a demand message to a thread manager and in which computer reading according to claim 13 is possible.
[Claim 15] The above-mentioned class is a record medium which is characterized by including the procedure to which an answerback message is made to output from a thread manager, and the procedure which receives the answerback message from a thread manager and in which computer reading according to claim 13 is possible.
[Claim 16] The above-mentioned class is a record medium which is characterized by including the procedure which deletes the content registered into the thread manager as a content processed by the thread and in which computer reading according to claim 13 is possible.

[Translation done.]

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention] This invention relates to the record medium

which recorded the operating system and in which computer reading is possible, and the record medium which recorded the application program executed using an operating system and in which computer reading is possible.

[0002]

[Description of the Prior Art] When realizing conventionally the application program containing a program module which receives a demand message and performs predetermined processing using a computer, the procedure which performs processing as shown in drawing 6 as a procedure of the receiving side of a demand message needed to be described.

[0003] That is, the main thread is first generated beforehand as a thread for receiving a demand message. This main thread generates the so-called message box, in order to receive a demand message, as shown in step S1. The demand message from the outside will be received and passed to this message box.

[0004] The main thread supervises the existence of the demand message received and passed to the message box. And as shown in step S2, the demand message concerned will be received, and if there is a demand message, as shown in step S3, a ** thread will be generated as a thread by which the processing based on the demand message concerned is made.

[0005] Thus, the generated ** thread performs predetermined processing according to a demand message, as shown in step S4. And when an answerback message needs to be returned as a result of processing predetermined [concerned], as shown in step S5, the answerback message concerned is transmitted to the main thread. Then, a ** thread is terminated as shown in step S6.

[0006] As mentioned above, when realizing an application program which

receives a demand message and performs predetermined processing, also until had described procedure in the detail about generation of a ** thread etc.

[0007]

[Problem(s) to be Solved by the Invention] By the way, in many application programs, a program module which receives a demand message and performs predetermined processing is used very frequently. And the conventional application program had described procedure which was mentioned above for such every program module each time. And in creation of an application program, the great effort was needed for description of such a procedure.

[0008] Moreover, in processing as shown in drawing 6, generation and termination of a ** thread will be repeated for every demand message. However, generation and termination of a thread are processing which needs many operations in comparison. For this reason, generation and termination of a thread had become the hindrance of the improvement in program execution effectiveness conventionally.

[0009] Moreover, when an activation resource is taken into consideration, I want to give an upper limit to the number of simultaneously generable threads in many cases, although a thread will be generated in processing as shown in drawing 6 whenever it receives a demand message. And when an upper limit needed to be given to the number of simultaneously generable conventionally threads, a still more complicated procedure needed to be described and the still greater effort was needed for programming.

[0010] This invention sets it as the main objects to offer the record medium which recorded the operating system which can describe easily the application program containing a program module which is proposed in view of the above conventional actual condition, receives a demand

message, and performs predetermined processing and in which computer reading is possible.

[0011]

[Means for Solving the Problem] As for the record medium which the 1st concerning this invention can computer read, it comes to record an operating system. And this operating system is equipped with the command which starts the thread manager which generates the thread which receives a demand message, and by which the processing based on the demand message concerned is made as an application program interface, the command which terminates a thread manager, and the command which registers into a thread manager the content processed by the thread.

[0012] Here, as for the above-mentioned operating system, it is desirable to have further the command to which an answerback message is made to output, the command which receives the answerback message from a thread manager, the command which deletes the content registered into the thread manager as a content processed by the thread as an application program interface from the command which delivers a demand message to a thread manager, and the thread manager.

[0013] Moreover, as for the record medium which the 2nd concerning this invention can computer read, it comes to record the operating system with which object-oriented was applied. And this operating system is equipped with the class including the procedure which starts the thread manager which generates the thread which receives a demand message, and by which the processing based on the demand message concerned is made as an application program interface, the procedure which terminates a thread manager, and the procedure which registers into a thread manager the content processed by the thread.

[0014] Here, as for the above-mentioned class, it is desirable to have

further the procedure which delivers a demand message to a thread manager, the procedure to which an answerback message is made to output from a thread manager, the procedure which receives the answerback message from a thread manager, the procedure which deletes the content registered into the thread manager as a content processed by the thread.

[0015] Moreover, it comes to record the application program with which the record medium which the 3rd concerning this invention can computer read is performed using an operating system. And this operating system is equipped with the command which starts the thread manager which generates the thread which receives a demand message, and by which the processing based on the demand message concerned is made as an application program interface, the command which terminates a thread manager, and the command which registers into a thread manager the content processed by the thread.

[0016] Here, as for the above-mentioned operating system, it is desirable to have further the command to which an answerback message is made to output, the command which receives the answerback message from a thread manager, the command which deletes the content registered into the thread manager as a content processed by the thread as an application program interface from the command which delivers a demand message to a thread manager, and the thread manager.

[0017] Moreover, it comes to record the application program with which the record medium which the 4th concerning this invention can computer read is performed using the operating system with which object-oriented was applied. And this operating system is equipped with the class including the procedure which starts the thread manager which generates the thread which receives a demand message, and by which the processing based on the demand message concerned is made as an

application program interface, the procedure which terminates a thread manager, and the procedure which registers into a thread manager the content processed by the thread.

[0018] Here, as for the above-mentioned class, it is desirable to have further the procedure which delivers a demand message to a thread manager, the procedure to which an answerback message is made to output from a thread manager, the procedure which receives the answerback message from a thread manager, the procedure which deletes the content registered into the thread manager as a content processed by the thread.

[0019]

[Embodiment of the Invention] Hereafter, the gestalt of operation of this invention is explained.

[0020] First, the example of 1 configuration of the computer system to which this invention is applied is explained.

[0021] This computer system A video tape recorder, a videodisk player, As it is included in AV (Audio and Visual) devices, such as an audio tape recorder or an audio disk player, the AV equipment concerned is controlled and it is shown in drawing 1 CPU4 connected to memory 2 and a bus 3 through the bridge 1 (Central Processing Unit), It has the serial port 6 connected to the bus 3 through the bridge 5, and the bus slot 7 connected to the bus 3, and transmission and reception of a signal are possible through a bus 3 among these.

[0022] CPU4 performs data processing and the AV equipment with which this computer system was incorporated is controlled based on data processing by this CPU4. A serial port 6 is connected to the controlled-system device 8 of the AV equipment with which this computer system was incorporated, and an exchange of a signal required for control is performed between a computer system and the

controlled- system device 8 through this serial port 6. The bus slot 7 is a slot used in case expansion is carried out, and when performing expansion of this computer system or an AV equipment, a device required for expansion can connect it to this bus slot 8.

[0023] Memory 2 has RAM (Random Access Memory)9 which is rewritable storage, and ROM (Read Only Memory)10 which is read- only storage, and it writes data in RAM10 through a bridge 1 if needed while data read CPU4 from RAM9 or ROM10 through a bridge 1 if needed. Here, the operating system which performs fundamental management of this computer system is saved beforehand at ROM10, and an application program required for control of an AV equipment etc. is saved at RAM9. However, you may make it also save beforehand an application program required for control of an AV equipment at ROM10.

[0024] Although the above is the fundamental configuration of a computer system, it cannot be overemphasized that you may have the device except having mentioned above the computer system to which this invention is applied. That is, the computer system to which this invention is applied may be equipped with devices for a communication link, such as a modem or a terminal adopter, may be equipped with external storage, such as a hard disk drive unit, flexible disk equipment, or a magnetic tape unit, may be equipped with input devices, such as a mouse and a keyboard, may be equipped with displays, such as a CRT display or a liquid crystal display, and may be equipped with output units, such as a printer or a plotter.

[0025] Moreover, although the above- mentioned computer system shall be included in an AV equipment, the application of this invention is not restricted to the computer system for AV equipment control, and it cannot be overemphasized that this invention is applicable to the computer system of various applications. However, since the efficient

activity of resources, such as memory, is attained according to this invention, when the limited resource is applied to a small computer system with required utilizing efficiently, especially this invention is effective.

[0026] And it is the record medium with which it comes to record the operating system used in order that the record medium which applied this invention, and in which computer reading is possible may start the above computer systems and in which computer reading is possible, and the record medium with which it comes to record the application program which uses the operating system concerned for a list and is executed and in which computer reading is possible, and, specifically, the above-mentioned memory 2 is equivalent to this.

[0027] Object-oriented is applied and this operating system (OS is called hereafter.) is equipped with some classes as an application program interface (API is called hereafter.).

[0028] And OS used in this invention is equipped with the class treating the thread manager (a dispatcher is called hereafter.) which generates the thread which receives a demand message, and by which the processing based on the demand message concerned is made as one of the classes which it has as API "Dispatcher." That is, in this invention, an element called a dispatcher is introduced as API.

[0029] The dispatcher is made as [receive / a demand message] and it can be considered at this point that it is a kind of a message box so that it may mention later. However, unlike the conventional message box, a dispatcher not only receives a demand message, but performs management of the thread for performing processing based on a demand message.

[0030] And this class "Dispatcher" is equipped with two or more procedure generally called a method in object-oriented as shown in

drawing 2. In addition, although a concrete name is attached and explained about a class, a method, etc. here, especially these names are not limited and are [in / with a natural thing / this invention] good as a name of arbitration.

[0031] As shown in drawing 2 , a class "Dispatcher" As a method, "public static Dispatcher Create (int maxThread)", "public void Destroy()" and "public void Send (Entry function, byte[] params, int paramSize)", "public Id Ask (Entry function, byte[] params, int paramSize)", "public void Reply (Object reply)", It has "public Object GetReply (Id id)", and "public Entry Register (Object action)" and "public void Unregister (Entry entry)."

[0032] Here, a method "public static Dispatcher Create (int maxThread)" is a procedure which starts a dispatcher. Here, an argument "maxThread" specifies the upper limit of the thread generated simultaneously. That is, the number of the threads generated by the dispatcher is made below into the number set as the argument "maxThread."

[0033] And if a method "public static Dispatcher Create (int maxThread)" is called by the program module of an application program, OS will start a dispatcher. At this time, the number of the threads simultaneously generated by the dispatcher concerned is made below into the number set as the argument "maxThread."

[0034] A method "public void Destroy()" is a procedure which terminates a dispatcher. That is, by the program module of an application program, if a method "public void Destroy()" is called, OS will perform the post process of a dispatcher.

[0035] A method "public void Send (Entry function, byte[] params, int paramSize)" is a procedure which delivers a demand message to a dispatcher. This method "public void Send (Entry function, byte[]

params, int paramSize)" is used when unnecessary in an answerback message.

[0036] this method "public void Send (Entry function, byte[] params, int paramSize)" -- setting -- an argument "function" -- this demand message -- the processing (action is called hereafter.) to call is specified. An argument "params" specifies a required argument, when calling the action concerned. An argument "paramSize" specifies the number of arguments "params."

[0037] And if a method "public void Send (Entry function, byte[] params, int paramSize)" is called by the program module of an application program, OS will call action specified by the argument "function", will deliver the argument specified as the action concerned by the argument "params", and will perform the action concerned.

[0038] A method "public Id Ask (Entry function, byte[] params, int paramSize)" is a procedure which delivers a demand message to a dispatcher. This method "public Id Ask (Entry function, byte[] params, int paramSize)" is used when an answerback message is required.

[0039] this method "public Id Ask (Entry function, byte[] params, int paramSize)" -- setting -- an argument "function" -- this demand -- a message -- **** -- action to call is specified. An argument "params" specifies a required argument, when calling the action concerned. An argument "paramSize" specifies the number of arguments "params."

[0040] And if a method "public Id Ask (Entry function, byte[] params, int paramSize)" is called by the program module of an application program, OS will call action specified by the argument "function", will deliver the argument specified as the action concerned by the argument "params", and will perform the action concerned.

[0041] A method "public void Reply (Object reply)" is a procedure to which an answerback message is made to output from a dispatcher, and

when returning an answerback message from a dispatcher, it is used. In this method "public void Reply (Object reply)", an argument "reply" specifies the answerback message from a dispatcher.

[0042] And if a method "public void Reply (Object reply)" is called by the program module of an application program, OS will output from a dispatcher the answerback message specified by the argument "reply."

[0043] A method "public Object GetReply (Id id)" is a procedure which receives the answerback message from a dispatcher, and when receiving the answerback message from a dispatcher, it is used. In this method "public Object GetReply (Id id)", an argument "id" specifies the return value from a method "public Id Ask (Entry function, byte[] params, int paramSize)."

[0044] And if a method "public Object GetReply (Id id)" is called by the program module of an application program, OS will make an answerback message the return value specified as the program module of an application program by the argument "id", and will be delivered.

[0045] A method "public Entry Register (Object action)" is a procedure which registers into a dispatcher action processed by the thread. In this method "publicEntry Register (Object action)", an argument "action" specifies action registered into a dispatcher.

[0046] In addition, the return value from this method "public Entry Register (Object action)" is used when calling action in a method "public void Send (Entry function, byte[] params, int paramSize)" and a method "public Id Ask (Entry function, byte[] params, int paramSize)."

[0047] And if a method "public Entry Register (Object action)" is called by the program module of an application program, OS will register action specified as the dispatcher by the argument "action."

[0048] A method "public void Unregister (Entry entry)" is a procedure which deletes action registered into the dispatcher as action processed

by the thread. In this method "public void Unregister (Entry entry)", an argument "entry" specifies action deleted from a dispatcher.

[0049] And if a method "public void Unregister (Entry entry)" is called by the program module of an application program, OS will delete action specified by the argument "entry" from a dispatcher.

[0050] By using the above classes "Dispatcher" as API, it becomes possible to describe very briefly a program module which receives a demand message and performs action, and, moreover, it becomes possible to realize the action concerned efficiently.

[0051] When using such a class "Dispatcher", as shown in drawing 3, starting of a dispatcher and registration of action are performed first. That is, in step S1-1, a dispatcher is first started using a method "public static Dispatcher Create (int maxThread)." Then, in step S1-2, a method "public Entry Register (Objectaction)" is used for the dispatcher concerned, and action is registered into it.

[0052] Then, action registered into the dispatcher is called and performed if needed by the method "public void Send (Entry function, byte[]params, int paramSize)" and the method "public Id Ask (Entry function, byte[]params, int paramSize)." Or using a method "public void Reply (Object reply)", an answerback message is made to output from a dispatcher and the answerback message concerned is received using a method "public Object GetReply (Id id)" if needed. Or action processed by the thread is further registered into a dispatcher using a method "public Entry Register (Object action)" if needed. Moreover, action registered into the dispatcher is deleted using a method "public void Unregister (Entry entry)" if needed. And if required processing is completed and a dispatcher becomes unnecessary, a dispatcher will be terminated using a method "public void Destroy()."

[0053] When the above classes "Dispatcher" are used as API, in a

program module, there is no need of describing generation and dissipation of a thread clearly. That is, since the need of performing generation and dissipation of a thread for every demand message is lost, in a program module side, the need of performing the generation of a thread and the management about dissipation which need many operations in comparison is lost. Therefore, when a demand message occurs in the program module of an application program, there is little processing taken to start action based on the demand message concerned, and it ends. Consequently, the processing effectiveness of a program module improves substantially.

[0054] About description of the program module when using the above classes "Dispatcher" as API, the concrete example is shown in drawing 4. In addition, the example shown in drawing 4 defines the class "Sample", using a class "Dispatcher" as API. And he is trying to register action "foo" and action "bar" into a dispatcher in the example shown in drawing 4.

[0055] In a program module as shown in drawing 4, when action "foo" and action "bar" are called, the thread for performing those actions is generated. However, the class "Dispatcher" is used as API here and it is managed by the dispatcher about generation and dissipation of a thread. Therefore, it is not necessary to describe generation and dissipation of a thread clearly, and in a program module side, as shown in drawing 4, description of a program module can be made very brief.

[0056] By the way, he is trying for an argument "maxThread" to prescribe the upper limit of the thread generated simultaneously by the method "public static Dispatcher Create (int maxThread)" mentioned above. And in a program module which receives a demand message and performs action, the number of the threads performed simultaneously can be easily restricted by using the class "Dispatcher" equipped with

such a method "public static Dispatcher Create (int maxThread)" as API.

[0057] Thus, the flow of processing when an argument "maxThread" prescribes a number of a thread of upper limits generated simultaneously is shown in drawing 5 .

[0058] As shown in drawing 5 , in step S2-1, a dispatcher is first started by the method "public static Dispatcher Create (int maxThread)." At this time, a number of threads specified by the argument "maxThread" are generated, and the queue of these threads is made.

[0059] Step S The dispatcher started by 2-1 supervises the existence of a demand message. And if a demand message is received as shown in step S2-2, a dispatcher will choose one thread from queues, as shown in step S2-3. Thus, in the selected thread, as shown in step S2-4, action demanded by the demand message is performed.

[0060] And in processing whose action concerned returns an answerback message, as shown in step S2-5, an answerback message is outputted as a result of the action concerned. And as shown in step S2-6, the thread which activation of action ended is returned to a queue.

[0061] Thus, since the need of performing generation and dissipation of a thread for every demand message will be lost if it is made to perform action which chose one thread from the queue and was demanded by the demand message by the thread concerned whenever it generates beforehand a number of threads specified by the argument "maxThread" and a dispatcher receives a demand message, it becomes possible to perform processing according to many demand messages very efficiently.

[0062] Moreover, it also becomes possible by setting an argument "maxThread" as "1" to use a dispatcher as a means of the mutual exclusion between the threads in programming using two or more threads. That is, those actions cease to be simultaneously performed by registering into the dispatcher of maxThread=1 action troubled if it

performs simultaneously, and being made to perform action.

[0063] In addition, although the example was given for OS which applied object-oriented in the above explanation, it is also possible to use OS which has not applied object-oriented in this invention. What is necessary is just to make OS equipped with the command which performs the same processing as the method mentioned by the above explanation as API, when using OS which has not applied object-oriented.

[0064] Namely, the command which starts a dispatcher and the command which terminates a dispatcher, The command which registers into a dispatcher the content processed by the thread, The command with which an answerback message delivers an unnecessary demand message to a dispatcher, The command with which an answerback message delivers a required demand message to a dispatcher, What is necessary is just to make OS equipped with the command to which an answerback message is made to output from a dispatcher, the command which receives the answerback message from a dispatcher, and the command which deletes the content registered into the dispatcher as a content processed by the thread as API.

[0065] Here, the command which starts a dispatcher is equivalent to an above-mentioned method "public static Dispatcher Create (int maxThread)." The command which terminates a dispatcher is equivalent to an above-mentioned method "public void Destroy()." The command which registers into a dispatcher the content processed by the thread is equivalent to an above-mentioned method "public Entry Register (Object action)."

[0066] Moreover, the command with which an answerback message delivers an unnecessary demand message to a dispatcher is equivalent to an above-mentioned method "public void Send (Entry function, byte[]

params, int paramSize)." The command with which an answerback message delivers a required demand message to a dispatcher is equivalent to an above-mentioned method "public Id Ask (Entry function, byte[] params, int paramSize)."

[0067] Moreover, the command to which an answerback message is made to output from a dispatcher is equivalent to an above-mentioned method "public void Reply (Object reply)." The command which receives the answerback message from a dispatcher is equivalent to an above-mentioned method "public Object GetReply (Id id)." The command which deletes the content registered into the dispatcher as a content processed by the thread is equivalent to an above-mentioned method "public void Unregister (Entry entry)."

[0068] In addition, in this invention, if an application program is performed using the above operating systems, especially the class, Field of application, etc. will not be limited. However, when an application program needs processing by two or more threads, especially this invention is effective.

[0069]

[Effect of the Invention] In case the application program containing a program module which receives a demand message and performs predetermined processing by applying this invention is created so that clearly from the above explanation, the description about generation, dissipation, etc. of a thread becomes unnecessary. Therefore, creation of an application program becomes very easy. And since the need of performing generation, dissipation, etc. of a thread for every demand message is lost, improvement in processing effectiveness can also be aimed at.

[Translation done.]

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-320216
(43)Date of publication of application : 04.12.1998

(51)Int.Cl. G06F 9/46

(21)Application number : 09-124442
(22)Date of filing : 14.05.1997

(71)Applicant : SONY CORP
(72)Inventor : KIKUCHI TOSHIKI
YOKOTE YASUHIKO

(54) COMPUTER READABLE RECORDING MEDIUM

(57)Abstract:
PROBLEM TO BE SOLVED: To easily describe an application program(AP) by providing an operating system(OS) with a command for controlling a sled managing program for generating a sled.
SOLUTION: The OS is provided with plural procedures called methods at a dispatcher for dealing with the sled managing program for generating the sled for accepting and processing a request message as one of classes of AP interface(API) while applying object orientation. Therefore, when using the dispatcher, the dispatcher is activated by using the prescribed method (S1-1). Afterwards, an action is registered by using the prescribed method for the dispatcher (S1-2). When required processing is completed, the dispatcher ends by using the prescribed method. Thus, a program module can be simply described.

LEGAL STATUS

[Date of request for examination]	30.01.2002
[Date of sending the examiner's decision of rejection]	13.12.2005
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]	
[Date of final disposal for application]	
[Patent number]	
[Date of registration]	
[Number of appeal against examiner's decision of rejection]	
[Date of requesting appeal against examiner's decision of rejection]	
[Date of extinction of right]	

(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（A）　(11)特許出願公開番号

特開平10－320216

(43)公開日　平成10年(1998)12月４日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　9/46	３４０	Ｇ０６Ｆ　9/46	３４０Ｂ

審査請求　未請求　請求項の数16　ＯＬ　（全９頁）

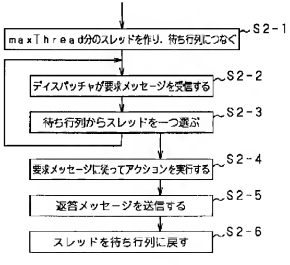
(21)出願番号　特願平9－124442

(22)出願日　平成９年(1997)５月14日

(71)出願人　000002185
ソニー株式会社
東京都品川区北品川６丁目７番35号
(72)発明者　菊地　俊樹
東京都品川区北品川６丁目７番35号　ソニー株式会社内
(72)発明者　横手　靖彦
東京都品川区北品川６丁目７番35号　ソニー株式会社内
(74)代理人　弁理士　小池　晃　（外２名）

(54)【発明の名称】　コンピュータ読み取り可能な記録媒体

(57)【要約】
【課題】　要求メッセージを受け取って所定の処理を行うようなプログラムモジュールを含むアプリケーションプログラムを容易に記述することが可能なオペレーティングシステムを提供する。
【解決手段】　アプリケーションインターフェースとして、要求メッセージを受け付けて当該要求メッセージに基づく処理がなされるスレッドを生成するスレッド管理プログラムを起動する手続きと、スレッド管理プログラムを終了させる手続きと、スレッドで処理される内容をスレッド管理プログラムに登録する手続きとを含むクラスを設ける。



1

【特許請求の範囲】

【請求項１】　アプリケーションプログラムインターフェースとして、

要求メッセージを受け付けて当該要求メッセージに基づく処理がなされるスレッドを生成するスレッド管理プログラムを起動するコマンドと、

スレッド管理プログラムを終了させるコマンドと、

スレッドで処理される内容をスレッド管理プログラムに登録するコマンドと、

を備えたオペレーティングシステムを記録したコンピュータ読み取り可能な記録媒体。

【請求項２】　上記オペレーティングシステムは、アプリケーションプログラムインターフェースとして、

スレッド管理プログラムに要求メッセージを受け渡すコマンドを備えていること

を特徴とする請求項１記載のコンピュータ読み取り可能な記録媒体。

【請求項３】　上記オペレーティングシステムは、アプリケーションプログラムインターフェースとして、

スレッド管理プログラムから返答メッセージを出力させるコマンドと、

スレッド管理プログラムからの返答メッセージを受け取るコマンドとを備えていることを特徴とする請求項１記載のコンピュータ読み取り可能な記録媒体。

【請求項４】　上記オペレーティングシステムは、アプリケーションプログラムインターフェースとして、

スレッドで処理される内容としてスレッド管理プログラムに登録された内容を削除するコマンドを備えていることを特徴とする請求項１記載のコンピュータ読み取り可能な記録媒体。

【請求項５】　要求メッセージを受け付けて当該要求メッセージに基づく処理がなされるスレッドを生成するスレッド管理プログラムを起動する手続きと、

スレッド管理プログラムを終了させる手続きと、

スレッドで処理される内容をスレッド管理プログラムに登録する手続きと、

を含むクラスをアプリケーションプログラムインターフェースとして備えたオペレーティングシステムを記録したコンピュータ読み取り可能な記録媒体。

【請求項６】　上記クラスは、スレッド管理プログラムに要求メッセージを受け渡す手続きを含むことを特徴とする請求項５記載のコンピュータ読み取り可能な記録媒体。

【請求項７】　上記クラスは、スレッド管理プログラムから返答メッセージを出力させる手続きと、スレッド管理プログラムからの返答メッセージを受け取る手続きとを含むことを特徴とする請求項５記載のコンピュータ読み取り可能な記録媒体。

【請求項８】　上記クラスは、スレッドで処理される内容としてスレッド管理プログラムに登録された内容を削除する手続きを含むことを特徴とする請求項５記載のコンピュータ読み取り可能な記録媒体。

2

【請求項９】　要求メッセージを受け付けて当該要求メッセージに基づく処理がなされるスレッドを生成するスレッド管理プログラムを起動するコマンドと、

スレッド管理プログラムを終了させるコマンドと、

スレッドで処理される内容をスレッド管理プログラムに登録するコマンドと、

をアプリケーションプログラムインターフェースとして備えたオペレーティングシステムを用いて実行されるアプリケーションプログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１０】　上記オペレーティングシステムは、アプリケーションプログラムインターフェースとして、スレッド管理プログラムに要求メッセージを受け渡すコマンドを備えていることを特徴とする請求項９記載のコンピュータ読み取り可能な記録媒体。

【請求項１１】　上記オペレーティングシステムは、アプリケーションプログラムインターフェースとして、

スレッド管理プログラムから返答メッセージを出力させるコマンドと、

スレッド管理プログラムからの返答メッセージを受け取るコマンドとを備えていることを特徴とする請求項９記載のコンピュータ読み取り可能な記録媒体。

【請求項１２】　上記オペレーティングシステムは、アプリケーションプログラムインターフェースとして、スレッドで処理される内容としてスレッド管理プログラムに登録された内容を削除するコマンドを備えていることを特徴とする請求項９記載のコンピュータ読み取り可能な記録媒体。

【請求項１３】　要求メッセージを受け付けて当該要求メッセージに基づく処理がなされるスレッドを生成するスレッド管理プログラムを起動する手続きと、

スレッド管理プログラムを終了させる手続きと、

スレッドで処理される内容をスレッド管理プログラムに登録する手続きと、

を含むクラスをアプリケーションプログラムインターフェースとして備えたオペレーティングシステムを用いて実行されるアプリケーションプログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１４】　上記クラスは、スレッド管理プログラムに要求メッセージを受け渡す手続きを含むことを特徴とする請求項１３記載のコンピュータ読み取り可能な記録媒体。

【請求項１５】　上記クラスは、スレッド管理プログラムから返答メッセージを出力させる手続きと、スレッド管理プログラムからの返答メッセージを受け取る手続きとを含むことを特徴とする請求項１３記載のコンピュータ読み取り可能な記録媒体。

【請求項１６】　上記クラスは、スレッドで処理される

内容としてスレッド管理プログラムに登録された内容を削除する手続きを含むことを特徴とする請求項１３記載のコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、オペレーティングシステムを記録したコンピュータ読み取り可能な記録媒体と、オペレーティングシステムを用いて実行されるアプリケーションプログラムを記録したコンピュータ読み取り可能な記録媒体とに関する。

【０００２】

【従来の技術】従来、コンピュータを用いて、要求メッセージを受け取って所定の処理を行うようなプログラムモジュールを含むアプリケーションプログラムを実現するときには、要求メッセージの受信側の手続きとして、図６に示すような処理を行う手続きを記述する必要があった。

【０００３】すなわち、先ず、要求メッセージを受け付けるためのスレッドとして、主スレッドを予め生成する。この主スレッドは、ステップＳ１に示すように、要求メッセージを受信するために、いわゆるメッセージボックスを生成する。外部からの要求メッセージは、このメッセージボックスに受け渡されることとなる。

【０００４】主スレッドは、メッセージボックスに受け渡された要求メッセージの有無を監視する。そして、要求メッセージがあれば、ステップＳ２に示すように、当該要求メッセージを受信し、ステップＳ３に示すように、当該要求メッセージに基づく処理がなされるスレッドとして、従スレッドを生成する。

【０００５】このように生成された従スレッドは、ステップＳ４に示すように、要求メッセージに従って所定の処理を行う。そして、当該所定の処理の結果として返答メッセージを返す必要がある場合は、ステップＳ５に示すように、当該返答メッセージを主スレッドへ送信する。その後、ステップＳ６に示すように、従スレッドを終了させる。

【０００６】以上のように、要求メッセージを受け取って所定の処理を行うようなアプリケーションプログラムを実現するときには、従スレッドの生成等についてまでも詳細に手続きを記述していた。

【０００７】

【発明が解決しようとする課題】ところで、多くのアプリケーションプログラムにおいて、要求メッセージを受け取って所定の処理を行うようなプログラムモジュールは、非常に頻繁に使用される。そして、従来のアプリケーションプログラムでは、このようなプログラムモジュール毎に、上述したような手続きを毎回記述していた。そして、アプリケーションプログラムの作成においては、このような手続きの記述のために、多大な労力を必要としていた。

【０００８】また、図６に示したような処理では、要求メッセージ毎に従スレッドの生成と終了を繰り返すこととなる。しかしながら、スレッドの生成や終了は、比較的に多くの演算を必要とする処理である。このため、従来は、スレッドの生成や終了が、プログラムの実行効率向上の妨げとなっていた。

【０００９】また、図６に示したような処理では、要求メッセージを受け付ける度にスレッドを生成することとなるが、実行資源を考慮した場合、同時に生成できるスレッドの数には上限をつけたいことが多い。そして、従来、同時に生成できるスレッドの数に上限をつける必要がある場合には、更に複雑な手続きを記述する必要があり、プログラミングに更に多大な労力を必要としていた。

【００１０】本発明は、以上のような従来の実情に鑑みて提案されたものであり、要求メッセージを受け取って所定の処理を行うようなプログラムモジュールを含むアプリケーションプログラムを、容易に記述することが可能なオペレーティングシステムを記録したコンピュータ読み取り可能な記録媒体を提供することを主な目的としている。

【００１１】

【課題を解決するための手段】本発明に係る第１のコンピュータ読み取り可能な記録媒体は、オペレーティングシステムが記録されてなる。そして、このオペレーティングシステムは、アプリケーションプログラムインターフェースとして、要求メッセージを受け付けて当該要求メッセージに基づく処理がなされるスレッドを生成するスレッド管理プログラムを起動するコマンドと、スレッド管理プログラムを終了させるコマンドと、スレッドで処理される内容をスレッド管理プログラムに登録するコマンドとを備えている。

【００１２】ここで、上記オペレーティングシステムは、アプリケーションプログラムインターフェースとして、更に、スレッド管理プログラムに要求メッセージを受け渡すコマンドや、スレッド管理プログラムから返答メッセージを出力させるコマンドや、スレッド管理プログラムからの返答メッセージを受け取るコマンドや、スレッドで処理される内容としてスレッド管理プログラムに登録された内容を削除するコマンド等を備えていることが好ましい。

【００１３】また、本発明に係る第２のコンピュータ読み取り可能な記録媒体は、オブジェクト指向が適用されたオペレーティングシステムが記録されてなる。そして、このオペレーティングシステムは、アプリケーションプログラムインターフェースとして、要求メッセージを受け付けて当該要求メッセージに基づく処理がなされるスレッドを生成するスレッド管理プログラムを起動する手続きと、スレッド管理プログラムを終了させる手続きと、スレッドで処理される内容をスレッド管理プログ

ラムに登録する手続きとを含むクラスを備えている。

【００１４】ここで、上記クラスは、更に、スレッド管理プログラムに要求メッセージを受け渡す手続きや、スレッド管理プログラムから返答メッセージを出力させる手続きや、スレッド管理プログラムからの返答メッセージを受け取る手続きや、スレッドで処理される内容としてスレッド管理プログラムに登録された内容を削除する手続き等を備えていることが好ましい。

【００１５】また、本発明に係る第３のコンピュータ読み取り可能な記録媒体は、オペレーティングシステムを用いて実行されるアプリケーションプログラムが記録されてなる。そして、このオペレーティングシステムは、アプリケーションプログラムインターフェースとして、要求メッセージを受け付けて当該要求メッセージに基づく処理がなされるスレッドを生成するスレッド管理プログラムを起動するコマンドと、スレッド管理プログラムを終了させるコマンドと、スレッドで処理される内容をスレッド管理プログラムに登録するコマンドとを備えている。

【００１６】ここで、上記オペレーティングシステムは、アプリケーションプログラムインターフェースとして、更に、スレッド管理プログラムに要求メッセージを受け渡すコマンドや、スレッド管理プログラムから返答メッセージを出力させるコマンドや、スレッド管理プログラムからの返答メッセージを受け取るコマンドや、スレッドで処理される内容としてスレッド管理プログラムに登録された内容を削除するコマンド等を備えていることが好ましい。

【００１７】また、本発明に係る第４のコンピュータ読み取り可能な記録媒体は、オブジェクト指向が適用されたオペレーティングシステムを用いて実行されるアプリケーションプログラムが記録されてなる。そして、このオペレーティングシステムは、アプリケーションプログラムインターフェースとして、要求メッセージを受け付けて当該要求メッセージに基づく処理がなされるスレッドを生成するスレッド管理プログラムを起動する手続きと、スレッド管理プログラムを終了させる手続きと、スレッドで処理される内容をスレッド管理プログラムに登録する手続きとを含むクラスを備えている。

【００１８】ここで、上記クラスは、更に、スレッド管理プログラムに要求メッセージを受け渡す手続きや、スレッド管理プログラムから返答メッセージを出力させる手続きや、スレッド管理プログラムからの返答メッセージを受け取る手続きや、スレッドで処理される内容としてスレッド管理プログラムに登録された内容を削除する手続き等を備えていることが好ましい。

【００１９】

【発明の実施の形態】以下、本発明の実施の形態について説明する。

【００２０】まず、本発明が適用されるコンピュータシステムの一構成例について説明する。

【００２１】このコンピュータシステムは、ビデオテープレコーダ、ビデオディスクプレーヤ、オーディオテープレコーダまたはオーディオディスクプレーヤ等のようなＡＶ（Audio and Visual）機器に組み込まれ、当該ＡＶ機器の制御を行うものであり、図１に示すように、ブリッジ１を介してメモリ２及びバス３に接続されたＣＰＵ（Central Processing Unit）４と、ブリッジ５を介してバス３に接続されたシリアルポート６と、バス３に接続されたバススロット７とを備えており、これらの間ではバス３を介して信号の送受信が可能となっている。

【００２２】ＣＰＵ４は、演算処理を行うものであり、このコンピュータシステムが組み込まれたＡＶ機器は、このＣＰＵ４による演算処理に基づいて制御される。シリアルポート６は、このコンピュータシステムが組み込まれたＡＶ機器の制御対象機器８に接続され、このシリアルポート６を介して、コンピュータシステムと制御対象機器８との間で、制御に必要な信号のやり取りが行われる。バススロット７は、機能拡張する際に使用されるスロットであり、このコンピュータシステム又はＡＶ機器の機能拡張を行うときに、このバススロット８に機能拡張に必要な機器が接続され得るようになっている。

【００２３】メモリ２は、書き換え可能な記憶装置であるＲＡＭ（Random Access Memory）９と、読み出し専用の記憶装置であるＲＯＭ（Read Only Memory）１０とを有しており、ＣＰＵ４は、必要に応じてＲＡＭ９又はＲＯＭ１０からブリッジ１を介してデータが読み出すとともに、必要に応じてブリッジ１を介してＲＡＭ１０にデータを書き込む。ここで、ＲＯＭ１０には、このコンピュータシステムの基本的な管理を行うオペレーティングシステム等が予め保存され、ＲＡＭ９には、ＡＶ機器の制御に必要なアプリケーションプログラム等が保存される。ただし、ＡＶ機器の制御に必要なアプリケーションプログラムもＲＯＭ１０に予め保存するようにしてもよい。

【００２４】以上がコンピュータシステムの基本的な構成であるが、本発明が適用されるコンピュータシステムは、上述した以外の機器を備えていてもよいことは言うまでもない。すなわち、本発明が適用されるコンピュータシステムは、例えば、モデム又はターミナルアダプタ等のような通信用機器を備えていても良いし、ハードディスク装置、フレキシブルディスク装置又は磁気テープ装置等のような外部記憶装置を備えていても良いし、マウスやキーボード等のような入力装置を備えていても良いし、ＣＲＴディスプレイ又は液晶ディスプレイ等のような表示装置を備えていても良いし、プリンタ又はプロッタ等のような出力装置を備えていても良い。

【００２５】また、上記コンピュータシステムは、ＡＶ機器に組み込まれるものとしたが、本発明の用途はＡＶ機器制御用のコンピュータシステムに限られるものでは

なく、本発明は様々な用途のコンピュータシステムに適用できることは言うまでもない。ただし、本発明によればメモリ等の資源の効率的な活用が可能となるので、限られた資源を効率的に活用することが必要な小型のコンピュータシステムに適用したときに、本発明は特に有効である。

【００２６】そして、本発明を適用したコンピュータ読み取り可能な記録媒体は、以上のようなコンピュータシステムを起動するために使用されるオペレーティングシステムが記録されてなるコンピュータ読み取り可能な記録媒体、並びに当該オペレーティングシステムを用いて実行されるアプリケーションプログラムが記録されてなるコンピュータ読み取り可能な記録媒体であり、具体的には、例えば上述のメモリ２がこれに相当する。

【００２７】このオペレーティングシステム（以下、ＯＳと称する。）は、オブジェクト指向が適用されており、アプリケーションプログラムインターフェース（以下、ＡＰＩと称する。）として、いくつかのクラスを備えている。

【００２８】そして、本発明において使用されるＯＳは、ＡＰＩとして備えているクラスの一つとして、要求メッセージを受け付けて当該要求メッセージに基づく処理がなされるスレッドを生成するスレッド管理プログラム（以下、ディスパッチャと称する。）を扱う、「Dispatcher」というクラスを備えている。すなわち、本発明では、ＡＰＩとして、ディスパッチャという要素を導入する。

【００２９】ディスパッチャは、後述するように、要求メッセージを受け付けるようになされており、この点では、メッセージボックスの一種と見なすことができる。ただし、ディスパッチャは、従来のメッセージボックスとは異なり、単に要求メッセージを受け付けるだけではなく、要求メッセージに基づく処理を行うためのスレッドの管理も行う。

【００３０】そして、このクラス「Dispatcher」は、図２に示すように、オブジェクト指向において一般にメソッドと称される手続きを複数備えている。なお、ここでは、クラスやメソッド等について、具体的な名称を付して説明するが、当然の事ながら、本発明においてこれらの名称は特に限定されるものではなく、任意の名称として良い。

【００３１】図２に示すように、クラス「Dispatcher」は、メソッドとして、「public static Dispatcher Create(int maxThread)」と、「public void Destroy()」と、「public void Send(Entry function,byte[] params,int paramSize)」と、「public Id Ask(Entry function,byte[] params,int paramSize)」と、「public void Reply(Object reply)」と、「public Object GetReply(Id id)」と、「public Entry Register(Object action)」と、「public void Unregister(Entry entry)」とを備えている。

【００３２】ここで、メソッド「public static Dispatcher Create(int maxThread)」は、ディスパッチャを起動する手続きである。ここで、引数「maxThread」は、同時に生成されるスレッドの上限を規定する。すなわち、ディスパッチャによって生成されるスレッドの数は、引数「maxThread」に設定された数以下とされる。

【００３３】そして、アプリケーションプログラムのプログラムモジュールによってメソッド「public static Dispatcher Create(int maxThread)」が呼び出されると、ＯＳはディスパッチャを起動する。このとき、当該ディスパッチャによって同時に生成されるスレッドの数は、引数「maxThread」に設定された数以下とする。

【００３４】メソッド「public void Destroy()」は、ディスパッチャを終了させる手続きである。すなわち、アプリケーションプログラムのプログラムモジュールによって、メソッド「public void Destroy()」が呼び出されると、ＯＳはディスパッチャの終了処理を行う。

【００３５】メソッド「public void Send(Entry function,byte[] params,int paramSize)」は、ディスパッチャに要求メッセージを受け渡す手続きである。このメソッド「public void Send(Entry function,byte[] params,int paramSize)」は、返答メッセージが必要ない場合に使用される。

【００３６】このメソッド「public void Send(Entry function,byte[] params,int paramSize)」において、引数「function」は、この要求メッセージによって呼び出す処理（以下、アクションと称する。）を指定する。引数「params」は、当該アクションを呼び出すときに必要な引数を指定する。引数「paramSize」は、引数「params」の数を指定する。

【００３７】そして、アプリケーションプログラムのプログラムモジュールによってメソッド「public void Send(Entry function,byte[] params,int paramSize)」が呼び出されると、ＯＳは、引数「function」で指定されたアクションを呼び出し、当該アクションに引数「params」で指定された引数を受け渡して、当該アクションを実行する。

【００３８】メソッド「public Id Ask(Entry function,byte[] params,int paramSize)」は、ディスパッチャに要求メッセージを受け渡す手続きである。このメソッド「public Id Ask(Entry function,byte[] params,int paramSize)」は、返答メッセージが必要な場合に使用される。

【００３９】このメソッド「public Id Ask(Entry function,byte[] params,int paramSize)」において、引数「function」は、この要求メッセージよって呼び出すアクションを指定する。引数「params」は、当該アクションを呼び出すときに必要な引数を指定する。引数「paramSize」は、引数「params」の数を指定する。

【００４０】そして、アプリケーションプログラムのプログラムモジュールによってメソッド「public Id Ask(Entry function,byte[] params,int paramSize)」が呼び出されると、ＯＳは、引数「function」で指定されたアクションを呼び出し、当該アクションに引数「params」で指定された引数を受け渡して、当該アクションを実行する。

【００４１】メソッド「public void Reply(Object reply)」は、ディスパッチャから返答メッセージを出力させる手続きであり、ディスパッチャから返答メッセージを返すときに使用される。このメソッド「public void Reply(Object reply)」において、引数「reply」は、ディスパッチャからの返答メッセージを指定する。

【００４２】そして、アプリケーションプログラムのプログラムモジュールによってメソッド「public void Reply(Object reply)」が呼び出されると、ＯＳは、引数「reply」で指定された返答メッセージをディスパッチャから出力する。

【００４３】メソッド「public Object GetReply(Id id)」は、ディスパッチャからの返答メッセージを受け取る手続きであり、ディスパッチャからの返答メッセージを受け取るときに使用される。このメソッド「public Object GetReply(Id id)」において、引数「id」は、メソッド「public Id Ask(Entry function,byte[] params,int paramSize)」からの戻り値を指定する。

【００４４】そして、アプリケーションプログラムのプログラムモジュールによってメソッド「public Object GetReply(Id id)」が呼び出されると、ＯＳは、アプリケーションプログラムのプログラムモジュールに、引数「id」で指定された戻り値を返答メッセージとして受け渡す。

【００４５】メソッド「public Entry Register(Object action)」は、スレッドで処理されるアクションをディスパッチャに登録する手続きである。このメソッド「publicEntry Register(Object action)」において、引数「action」は、ディスパッチャに登録するアクションを指定する。

【００４６】なお、このメソッド「public Entry Register(Object action)」からの戻り値は、メソッド「public void Send(Entry function,byte[] params,int paramSize)」及びメソッド「public Id Ask(Entry function,byte[] params,int paramSize)」においてアクションを呼び出すときに使用される。

【００４７】そして、アプリケーションプログラムのプログラムモジュールによってメソッド「public Entry Register(Object action)」が呼び出されると、ＯＳは、ディスパッチャに、引数「action」で指定されたアクションを登録する。

【００４８】メソッド「public void Unregister(Entry entry)」は、スレッドで処理されるアクションとしてディスパッチャに登録されたアクションを削除する手続きである。このメソッド「public void Unregister(Entry entry)」において、引数「entry」は、ディスパッチャから削除するアクションを指定する。

【００４９】そして、アプリケーションプログラムのプログラムモジュールによってメソッド「public void Unregister(Entry entry)」が呼び出されると、ＯＳは、引数「entry」で指定されたアクションを、ディスパッチャから削除する。

【００５０】以上のようなクラス「Dispatcher」をＡＰＩとして使用することにより、要求メッセージを受けとってアクションを行うようなプログラムモジュールを非常に簡潔に記述することが可能となり、しかも、当該アクションを効率良く実現することが可能になる。

【００５１】このようなクラス「Dispatcher」を用いるときは、例えば、先ず、図３に示すように、ディスパッチャの起動と、アクションの登録とを行う。すなわち、先ず、ステップＳ１－１において、メソッド「public static Dispatcher Create(int maxThread)」を用いて、ディスパッチャを起動する。その後、ステップＳ１－２において、当該ディスパッチャに、メソッド「public Entry Register(Objectaction)」を用いて、アクションを登録する。

【００５２】その後、必要に応じて、メソッド「public void Send(Entry function,byte[]params,int paramSize)」やメソッド「public Id Ask(Entry function,byte[]params,int paramSize)」により、ディスパッチャに登録されたアクションを呼び出して実行する。或いは、必要に応じて、メソッド「public void Reply(Object reply)」を用いて、ディスパッチャから返答メッセージを出力させ、当該返答メッセージを、メソッド「public Object GetReply(Id id)」を用いて受け取るようにする。或いは、必要に応じて、メソッド「public Entry Register(Object action)」を用いて、スレッドで処理されるアクションを更にディスパッチャに登録する。また、必要に応じて、メソッド「public void Unregister(Entry entry)」を用いて、ディスパッチャに登録されているアクションを削除する。そして、必要な処理が完了して、ディスパッチャが不要になったら、メソッド「public void Destroy()」を用いて、ディスパッチャを終了させる。

【００５３】以上のようなクラス「Dispatcher」をＡＰＩとして用いた場合、プログラムモジュールにおいて、スレッドの生成や消滅について明示的に記述する必要が無い。すなわち、要求メッセージ毎にスレッドの生成や消滅を行う必要が無くなるので、プログラムモジュールの側では、比較的に多くの演算を必要とする、スレッドの生成や消滅についての管理を行う必要が無くなる。したがって、アプリケーションプログラムのプログラムモジュールにおいて要求メッセージが発生したときに、当

該要求メッセージに基づくアクションが開始されるまでに要する処理が少なくて済む。この結果、プログラムモジュールの処理効率が大幅に向上する。

【００５４】以上のようなクラス「Dispatcher」をＡＰＩとして用いたときのプログラムモジュールの記述について、その具体的な一例を図４に示す。なお、図４に示した例では、クラス「Dispatcher」をＡＰＩとして用いて、クラス「Sample」を定義している。そして、図４に示した例では、アクション「foo」と、アクション「bar」とを、ディスパッチャに登録するようにしている。

【００５５】図４に示したようなプログラムモジュールにおいて、アクション「foo」やアクション「bar」が呼び出されるときには、それらのアクションを実行するためのスレッドが生成される。しかしながら、ここではクラス「Dispatcher」をＡＰＩとして用いており、スレッドの生成や消滅については、ディスパッチャによって管理される。したがって、プログラムモジュールの側では、スレッドの生成や消滅について明示的に記述する必要がなく、図４に示したように、プログラムモジュールの記述を非常に簡潔なものとすることができる。

【００５６】ところで、上述したメソッド「public static Dispatcher Create(int maxThread)」では、引数「maxThread」により、同時に生成されるスレッドの上限を規定するようにしている。そして、このようなメソッド「public static Dispatcher Create(int maxThread)」を備えたクラス「Dispatcher」をＡＰＩとして使用することにより、要求メッセージを受け取ってアクションを行うようなプログラムモジュールにおいて、同時に実行されるスレッドの数を、容易に制限することができる。

【００５７】このように、同時に生成されるスレッドの数の上限を、引数「maxThread」により規定するようにしたときの処理の流れを図５に示す。

【００５８】図５に示すように、先ず、ステップＳ２－１において、メソッド「public static Dispatcher Create(int maxThread)」により、ディスパッチャを起動する。このとき、引数「maxThread」で規定された数のスレッドが生成され、これらのスレッドの待ち行列が作られる。

【００５９】ステップＳ２－１で起動されたディスパッチャは、要求メッセージの有無を監視する。そして、ディスパッチャは、ステップＳ２－２に示すように、要求メッセージを受信したら、ステップＳ２－３に示すように、待ち行列の中からスレッドを一つ選択する。このように選択されたスレッドにおいて、ステップＳ２－４に示すように、要求メッセージで要求されたアクションが行われる。

【００６０】そして、当該アクションが返答メッセージを返すような処理の場合には、ステップＳ２－５に示すように、当該アクションの結果として、返答メッセージが出力される。そして、ステップＳ２－６に示すように、アクションの実行が終了したスレッドは、待ち行列へと戻される。

【００６１】このように、引数「maxThread」で規定された数のスレッドを予め生成しておき、ディスパッチャが要求メッセージを受信する毎に待ち行列からスレッドを一つ選択して、当該スレッドで要求メッセージで要求されたアクションを行うようにすれば、要求メッセージ毎にスレッドの生成や消滅を行う必要が無くなるので、多数の要求メッセージに応じた処理を、非常に効率良く行うことが可能となる。

【００６２】また、引数「maxThread」を「１」に設定することにより、複数のスレッドを用いるプログラミングにおけるスレッド間の相互排他の手段として、ディスパッチャを使用することも可能となる。つまり、同時に実行されては困るアクションは、maxThread＝１のディスパッチャに登録してアクションを行うようにすることにより、それらのアクションが同時には実行されないようになる。

【００６３】なお、以上の説明では、オブジェクト指向を適用したＯＳを例を挙げたが、本発明では、オブジェクト指向を適用していないＯＳを使用することも可能である。オブジェクト指向を適用していないＯＳを用いるときは、以上の説明で挙げたメソッドと同様な処理を行うコマンドを、ＡＰＩとしてＯＳに備えさせればよい。

【００６４】すなわち、ディスパッチャを起動するコマンドと、ディスパッチャを終了させるコマンドと、スレッドで処理される内容をディスパッチャに登録するコマンドと、返答メッセージが不要な要求メッセージをディスパッチャに受け渡すコマンドと、返答メッセージが必要な要求メッセージをディスパッチャに受け渡すコマンドと、ディスパッチャから返答メッセージを出力させるコマンドと、ディスパッチャからの返答メッセージを受け取るコマンドと、スレッドで処理される内容としてディスパッチャに登録された内容を削除するコマンドとを、ＡＰＩとしてＯＳに備えさせればよい。

【００６５】ここで、ディスパッチャを起動するコマンドは、上述のメソッド「public static Dispatcher Create(int maxThread)」に相当する。ディスパッチャを終了させるコマンドは、上述のメソッド「public void Destroy()」に相当する。スレッドで処理される内容をディスパッチャに登録するコマンドは、上述のメソッド「public Entry Register(Object action)」に相当する。

【００６６】また、返答メッセージが不要な要求メッセージをディスパッチャに受け渡すコマンドは、上述のメソッド「public void Send(Entry function,byte□ params,int paramSize)」に相当する。返答メッセージが必要な要求メッセージをディスパッチャに受け渡すコマンドは、上述のメソッド「public Id Ask(Entry functio

n,byte□ params,int paramSize)」に相当する。

【００６７】また、ディスパッチャから返答メッセージを出力させるコマンドは、上述のメソッド「public void Reply(Object reply)」に相当する。ディスパッチャからの返答メッセージを受け取るコマンドは、上述のメソッド「public Object GetReply(Id id)」に相当する。スレッドで処理される内容としてディスパッチャに登録された内容を削除するコマンドは、上述のメソッド「public void Unregister(Entry entry)」に相当する。

【００６８】なお、本発明において、アプリケーションプログラムは、以上のようなオペレーティングシステムを用いて実行されるものであるならば、その種類や適用分野等は、特に限定されるものでない。ただし、アプリケーションプログラムが、複数のスレッドでの処理を必要とするようなときに、本発明は特に有効である。

【００６９】

【発明の効果】以上の説明から明らかなように、本発明を適用することにより、要求メッセージを受け取って所定の処理を行うようなプログラムモジュールを含むアプリケーションプログラムを作成する際に、スレッドの生成や消滅等についての記述が不要となる。したがって、アプリケーションプログラムの作成が非常に容易にな＊る。しかも、要求メッセージ毎にスレッドの生成や消滅等を行う必要が無くなるので、処理効率の向上を図ることもできる。

【図面の簡単な説明】

【図１】本発明が適用されるコンピュータシステムの一例を示すブロック図である。

【図２】クラス「Dispatcher」の構造を示す図である。

【図３】ディスパッチャを用いるときの処理の流れを示す図である。

【図４】プログラムモジュールの記述の一例を示す図である。

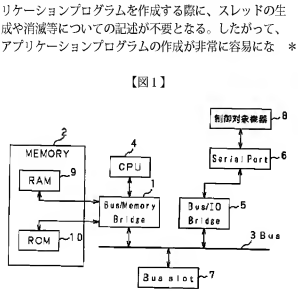
【図５】ディスパッチャを用いて、同時に生成されるスレッドの数の上限を規定したときの処理の流れを示す図である。

【図６】要求メッセージを受け取って所定の処理を行うようなプログラムモジュールについて、従来の処理の流れを示す図である。

【符号の説明】

1　ブリッジ、　2　メモリ、　3　バス、　4　ＣＰＵ、　5　ブリッジ、６　シリアルポート、　7　バススロット、　8　制御対象機器、　９　ＲＡＭ、　１０　ＲＯＭ

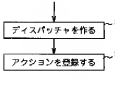
【図１】



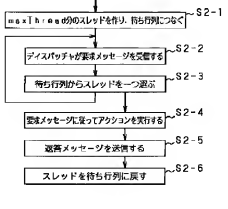
【図２】

```
public final class Dispatcher {
    public static Dispatcher Create(int maxThread);
    public void Destroy();
    public void Send(Entry function,byte[] params,int paramSize);
    public Id Ask(Entry function,byte[] params,int paramSize);
    public void Reply(Object reply);
    public Object GetReply(Id id);
    public Entry Register(Object action);
    public void Unregister(Entry entry);
}
```

【図３】



【図５】



【図６】

コンピュータシステムの一例

【図４】

```
public class Sample {
    private Dispatcher despatch;
    private Entry fooEntry;
    private Entry barEntry;
    public void foo(byte[] msg,int size) {
        アクション「foo」の内容
        返答メッセージ「Reply」の作成
        despatch.Reply(reply);
    }
    public void bar(byte[] msg,int size) {
        アクション「bar」の内容
    }
    public static void main(String[] args) {
        dispatch=Dispatcher:Create(1);
        fooEntry=despatch.Regester(this.foo);
        barEntry=despatch.Regester(this.bar);
    }
}
```

【図６】